Quantum Matcha Tea

An efficient matrix product state simulator for quantum circuits

Marco Ballarin
Università degli studi di Padova
Running quantum algorithms
Running quantum algorithms

+ Real hardware
- Noisy
- Limited number of qubits
Running quantum algorithms

- Real hardware
  - Noisy
  - Limited number of qubits

+ High # of qubits +
  - Flexibility (observables)
  - Depth of the circuit

- Noisy
- Limited number of qubits

Quantum hardware

cuQuantum

Quantum algorithm
Running quantum algorithms

Quantum hardware
- + Real hardware
- - Noisy
- - Limited number of qubits

Exact simulator
- + Access to exact state
- - Limited number of qubits

Quantum algorithm

High # of qubits +
- Flexibility (observables)
- - Depth of the circuit -

cuQuantum
- + Real hardware
- - Noisy
- - Limited number of qubits

- + Access to exact state
- - Limited number of qubits

Real hardware

Noisy
Limited number of qubits
Running quantum algorithms

Quantum hardware
- + Real hardware
- - Noisy
- - Limited number of qubits

Quantum algorithm
- + Access to exact state
- - Limited number of qubits

Exact simulator

High # of qubits +
Flexibility (observables) -
Depth of the circuit -

cuQuantum

High # of qubits +
Flexibility (# of T gates) -

Clifford simulator

Real hardware
- - Noisy
- - Limited number of qubits

Exact simulator

High # of qubits +
Flexibility (# of T gates) -
Running quantum algorithms

- + Real hardware
  - Noisy
  - Limited number of qubits

- + Access to exact state
  - Limited number of qubits

- + High # of qubits
  - Flexibility (entanglement)

- - Flexible hardware
- - Noisy
- - Limited number of qubits

- + High # of qubits
  - Flexibility (observables)
  - Depth of the circuit

- + Access to exact state

- + Flexibility (# of T gates)

- - Limited number of qubits
Why tensor networks

\[ \text{dim}(\mathcal{H}) = 2^n \]

We can represent a subset efficiently

?
Why tensor networks

\[ \dim(\mathcal{H}) = 2^n \]

We can represent a subset efficiently
Why tensor networks

\[ \dim(\mathcal{H}) = 2^n \]

\[ |\psi\rangle = \sum_{\alpha=1}^{\chi} \lambda_\alpha |A_\alpha\rangle |B_\alpha\rangle \]

We can represent a subset efficiently.
Why tensor networks

\[ \dim(\mathcal{H}) = 2^n \]

We can represent a subset efficiently

\[ |\psi\rangle = \sum_{\alpha=1}^{\chi} \lambda_\alpha |A_\alpha\rangle |B_\alpha\rangle \]

Tensor networks compress the quantum correlations between subsystems ⇒ **compress entanglement**
Why tensor networks

\[ \dim(\mathcal{H}) = 2^n \]

We can represent a subset efficiently

\[ |\psi\rangle = \sum_{\alpha=1}^{\chi} \lambda_\alpha |A_\alpha\rangle |B_\alpha\rangle \]

Tensor networks compress the quantum correlations between subsystems \( \Rightarrow \text{compress entanglement} \)

Only keep highest \( \chi \) Schmidt values
Matrix product states

\[ O(2^n) \rightarrow O(2n\chi^2) \]
Matrix product states

Each tensor (ball) encodes the state of a qubit

\[ O(2^n) \rightarrow O(2n\chi^2) \]

Memory requirements
Matrix product states

Each tensor (ball) encodes the state of a qubit

Bonds encode entanglement between qubits

Memory requirements

\[ O(2^n) \rightarrow O(2n\chi^2) \]
Matrix product states

Each tensor (ball) encodes the state of a qubit

Bonds encode entanglement between qubits

Memory requirements

$O(2^n) \rightarrow O(2n\chi^2)$
Matrix product states

Each tensor (ball) encodes the state of a qubit

Bonds encode entanglement between qubits

Memory requirements

\[ O(2^n) \rightarrow O(2n\chi^2) \]

MPS simulations are not limited by the number of qubits but by the entanglement.
Quantum TEA distribution

Tensor network \(\textbf{T}E\textbf{A}\) Applications

Emulator
Quantum TEA distribution

Tensor network ➔ **T** ➔ **E** ➔ **A** ➔ Applications

- **Quantum tea leaves:** *Utility*
- **Quantum matcha tea:** *quantum circuit HPC simulations*
- **Quantum red tea:** *tensor handling*
- **Quantum chai tea:** *AI and ML with tensor networks*
- **Quantum green tea:** *Schrödinger equation solution for many-body states*
Quantum TEA distribution

- **Tensor network**
- **Emulator**
- **Applications**

Quantum tea leaves: **Utility**

Quantum matcha tea: **quantum circuit HPC simulations**

Quantum red tea: **tensor handling**

Quantum chai tea: **AI and ML with tensor networks**

Quantum green tea: **Schrödinger equation solution for many-body states**
Quantum Matcha Tea workflow

- Quantum circuit
- Observables
- Python interface, definition of the problem
Quantum Matcha Tea workflow

Quantum circuit
Observables
Python interface, definition of the problem

Matrix product state simulator
Quantum Matcha Tea workflow

Quantum circuit

Observables

Python interface, definition of the problem

Matrix product state simulator

Backends for running the simulations

Serial CPU
Multinode MPI CPU
Serial GPU

NumPy
CuPy
F

Matrix product state simulator
Quantum Matcha Tea workflow

Quantum circuit
  ➕ ➕
  Observables

Python interface, definition of the problem

Matrix product state simulator

Backends for running the simulations
  ➕ ➕ ➕
  NumPy
  CuPy
  F

Serial CPU
Multinode MPI CPU
Serial GPU

Not public yet
Quantum Matcha Tea workflow

Quantum circuit
Observables
Python interface, definition of the problem

Matrix product state simulator

Observables
Runtime statistics
Convergence checks
Python interface output

Backends for running the simulations
NumPy
CuPy
Serial CPU
Multinode MPI CPU
Serial GPU

Not public yet
Convergence checks & error bound

$$|\psi\rangle = \sum_{\alpha=1}^{\chi_T^{i-1}} |A_\alpha\rangle \lambda_\alpha |B_\alpha\rangle$$
Convergence checks & error bound

\[ |\psi\rangle = \sum_{\alpha=1}^{\chi_T^{i-1}} \lambda_\alpha |A_\alpha\rangle |B_\alpha\rangle \]
Convergence checks & error bound

\[ |\psi\rangle = \sum_{\alpha=1}^{\chi_T^{i-1}} \alpha |A_\alpha\rangle \lambda_\alpha |B_\alpha\rangle \]
Convergence checks & error bound

\[ |\psi\rangle = \sum_{\alpha=1}^{\chi_T} \chi^{i-1}_T |A_{\alpha}\rangle \lambda_{\alpha} |B_{\alpha}\rangle \]

Only keep highest \( \chi \) singular values, \( |\phi\rangle \)
Convergence checks & error bound

\[ |\psi\rangle = \sum_{\alpha=1}^{\chi_T} \lambda_\alpha^{i} |A_\alpha\rangle \]

Only keep highest \( \chi \) singular values, \( |\phi\rangle \)
Convergence checks & error bound

\[ | \psi \rangle = \sum_{\alpha=1}^{\chi T} |A_\alpha \rangle \lambda_\alpha |B_\alpha \rangle \]

Only keep highest \( \chi \) singular values, \( | \phi \rangle \)

Fidelity of the state

\[ \mathcal{F}_i(\chi) = \left| \langle \psi \mid \phi \rangle \right|^2 = \left| 1 - \sum_{\alpha=\chi+1}^{\chi T} \lambda_\alpha^2 \right|^2 \]
Convergence checks & error bound

\[ |\psi\rangle = \sum_{\alpha=1}^{\chi_T} |\psi\rangle |A_\alpha\rangle |B_\alpha\rangle \]

Only keep highest \( \chi \) singular values, \( |\phi\rangle \)

Fidelity of the state

\[ \mathcal{F}_i(\chi) = \left| \langle \psi | \phi \rangle \right|^2 = 1 - \sum_{\alpha=\chi+1}^{\chi_T} \lambda_{\alpha}^2 \]

Computed during the simulation
Convergence and error checks
Convergence and error checks

Fidelity of the state after a single gate

\[ \mathcal{F}_i(\chi) = 1 - \sum_{\alpha=\chi+1}^{\chi_T} \lambda_\alpha^2 \]

\[ = 1 - \sum_{\alpha=\chi+1}^{\chi_T} \lambda_\alpha^2 \]
Fidelity of the state after a single gate

$$\mathcal{F}_i(\chi) = 1 - \sum_{\alpha=\chi+1}^{\chi_T} \lambda_{\alpha}^2$$

Fidelity at the end of the simulation

$$\mathcal{F}_{tot}(\chi) \geq \prod_{i} \mathcal{F}_i(\chi)$$
Gates acts on the same qubits: we contract gates together and only after with state.
Gates acts on the same qubits: we contract gates together and only after with state.
Optimisation & parallelism

Gates acts on the same qubits: we contract gates together and only after with state
Optimisation & parallelism

Gates acts on the same qubits: we contract gates together and only after with state.
Optimisation & parallelism

Gates acts on the same qubits: we contract gates together and only after with state
Optimisation & parallelism

Gates acts on the same qubits: we contract gates together and only after with state.

Copy of the qubit state

Node 0

Node 1
Optimisation & parallelism

Gates acts on the same qubits: we contract gates together and only after with state.

Copy of the qubit state

Node 0

Node 1

Barrier to wait for the data from node 0
Optimisation & parallelism

Gates acts on the same qubits: we contract gates together and only after with state.

Copy of the qubit state

Node 0

Node 1

Barrier to wait for the data from node 0

A GOOD PARALLEL SCALING INCREASES ERRORS DUE TO AN ALGORITHMIC SUBTLETY
Benchmarks

QFT

Entangling block size

Computational time [s]

$n = 100, \chi_{\text{max}} = 1024, 16$ threads

- qiskit
- fortran
- numpy

Wall time
Benchmarks

![Diagram of random entangling blocks](image)

Strongly entangling

<table>
<thead>
<tr>
<th>Entangling block size</th>
<th>Computational time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 100, \chi_{max} = 1024, 16$ threads</td>
<td></td>
</tr>
</tbody>
</table>

- qiskit
- fortran
- numpy
- Wall time

Graph showing computational time vs. entangling block size.
Benchmarks

Strongly entangling

$\chi_{max}$ reached

$n = 100, \chi_{max} = 1024, 16$ threads

Run on Galileo100

Computational time [s]

Entangling block size

qiskit

fortran

numpy

Wall time
Applications

**Entanglement entropy production in QNN**
- Simulations up to 50 qubits
- Bond dimension of 4096
- 11h of runtime on Galileo100

**Ab initio two-dimensional digital twin for quantum computer**
- Use of the unbiased sampling
- Quantum matcha tea simulations used as target state to compute the fidelity of a simulation with crosstalk
Conclusions

MPS simulations are not limited by the number of qubits but by the entanglement.
Conclusions

MPS simulations are not limited by the number of qubits but by the entanglement.

Easy-to-use python frontend and fast HPC-ready backend (Both GPU and CPU)

\[
\begin{align*}
\text{Computational time [s]} & = 10^n \\
\text{Entangling block size} & = n = 100, \chi_{\text{max}} = 1024, 16 \text{ threads}
\end{align*}
\]
Conclusions

MPS simulations are not limited by the number of qubits but by the entanglement.

Easy-to-use python frontend and fast HPC-ready backend (Both GPU and CPU)

Error analysis tools and efficient computations of observables optimised for the MPS representation.

\[ n = 100, \chi_{\text{max}} = 1024, 16 \text{ threads} \]
Thanks for your attention
Efficient sampling of final state

![Graph showing state coverage over number of samples]
Efficient sampling of final state

Reuse the computation of this node

Sample random number $n = 0.05, 0.29$

We know which states we did not sample and can sample only here in second round